How to call erase with a reverse iterator

I am trying to do something like this:

```
for ( std::list< Cursor::Enum >::reverse_iterator i = m_CursorStack.rbegin(); i !=
m_CursorStack.rend(); ++i )
{
    if ( *i == pCursor )
    {
        m_CursorStack.erase( i );
        break;
    }
}
```

However erase takes an iterator and not a reverse iterator. is there a way to convert a reverse iterator to a regular iterator or another way to remove this element from the list?

c++

asked Dec 2 '09 at 1:41
0xC0DEFACE
3,316 3 20 31

10 As an aside, when writing loops like these, don't repeatedly compute the end iterator as you do here with `i != m_CursorStack.rend()` . Instead, write `i = m_CursorStack.rbegin(), end = m_CursorStack.rend(); i != end;` . That is, initialize an iterator you can keep around for repeated comparison -- assuming that the end position won't be changing as a side effect of your loop body. – seh Dec 2 '09 at 3:10

It seems to me that the obvious question here would be why you're doing this at all. What do you gain from traversing the list in reverse? What do you gain by writing this code yourself instead of using `std::remove` ? – Jerry Coffin Dec 2 '09 at 3:32

And is an iterator on a std::list still valid to increment after the element to which it refers has been erased? – Steve Jessop Dec 2 '09 at 3:45

3 I only want to remove 1 element, thus the 'break;', using 'remove' would get rid of any that match taking longer and not doing what i want. The element I want to remove in this particular case will nearly always be the end of the list or very close to it, so iterating in reverse is also quicker and better suited to the problem. – 0xC0DEFACE Dec 2 '09 at 3:51

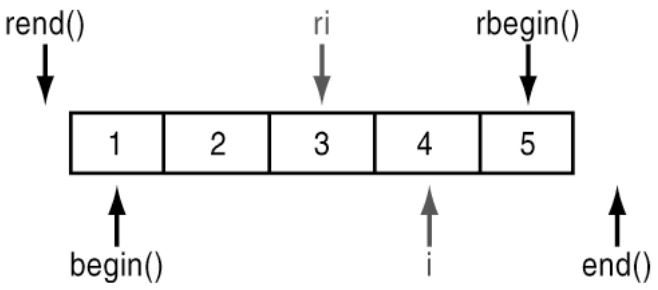
@Steve - It doesn't get incremented cause when its erased it breaks out of the loop – 0xC0DEFACE Dec 2 '09 at 3:53

7 Answers

After some more research and testing I found the solution. Apparently according to the standard [24.4.1/1] the relationship between i.base() and i is:

```
&*(reverse_iterator(i)) == &*(i - 1)
```

(from a Dr. Dobbs article):



So you need to apply an offset when getting the base(). Therefore the solution is:

```
m_CursorStack.erase( --(i.base()) );
```

EDIT

Updating for C++11.

reverse_iterator `i` is unchanged:

```
m_CursorStack.erase( std::next(i).base() );
```

reverse_iterator `i` is advanced:

```
std::advance(i, 1);
m_CursorStack.erase( i.base() );
```

I find this much clearer than my previous solution. Use whichever you require.

edited Jul 10 '15 at 6:31 answered Dec 2 '09 at 2:09
0xC0DEFACE
3,316 3 20 31

Thanks for pointing out that erase(i.base()) is wrong -- I have some code that does that and works, and now I don't know why! :(– Dan Dec 2 '09 at 4:31

22 You should take note of a bit more of the article you cited - to be portable the expression should be `m_CursorStack.erase( (++i).base())` (man, doing this stuff with reverse iterators makes my head hurt...). It should also be noted that the DDJ article is incorporated into Meyer's "Effective STL" book. – Michael Burr Dec 2 '09 at 6:25

4 I find that diagram more confusing than helpful. Since rbegin, ri and rend are all *actually* pointing at the element to the right of what they are drawn to be pointing at. The diagram shows what element you would access if you `*` them, but we're talking about what element you'd be pointing at if you `base` them, which is one element to the right. I'm not such a fan of the `--(i.base())` or `(++i).base()` solutions since they mutate the iterator. I prefer `(i+1).base()` which works as well. – mgiuca Jun 16 '12 at 11:20

2 Reverse iterators are liars.. when deferenced, a reverse iterator returns the element *before it*. See here – bobobobo Feb 7 '13 at 20:51

1 Just to be absolutely clear, this technique still cannot be used in a normal for loop (where the iterator is incremented in the normal manner). See stackoverflow.com/questions/37005449/… – logidelic May 3 at 18:02

Microsoft
Microsoft Azure
Got 5 minutes? Then you have time
to deploy your first cloud solution.
Try Azure—free

Please note that `m_CursorStack.erase( (++i).base())` may be a problem if used in a `for` loop (see original question) because it changes the value of i. Correct expression is `m_CursorStack.erase((i+1).base())`

answered Sep 28 '10 at 0:33
Andrey
111 1 2

3 You'd need to create a copy of the iterator and do `iterator j = i ; ++j` , because `i+1` doesn't work on an iterator, but that's the right idea – bobobobo Feb 7 '13 at 21:12

1 @bobobobo, you can use `m_CursorStack.erase(boost::next(i).base())` with Boost. or in C++11 `m_CursorStack.erase(std::next(i).base())` – alfC Jun 10 '14 at 8:59

> ... or another way to remove this element from the list?

This requires the `-std=c++11` flag (for `auto`):

```
auto it=vt.end();
while (it>vt.begin())
{
    it--;
    if (*it == pCursor) //{ delete *it;
        it = vt.erase(it); //}
}
```

answered Dec 10 '12 at 15:35
slashmais
4,396 7 38 60

While using the `reverse_iterator` 's `base()` method and decrementing the result works here, it's worth noting that `reverse_iterator` s are not given the same status as regular `iterator` s. In general, you should prefer regular `iterator` s to `reverse_iterator` s (as well as to `const_iterator` s and `const_reverse_iterator` s), for precisely reasons like this. See Doctor Dobbs' Journal for an in-depth discussion of why.

answered Dec 2 '09 at 2:19
Adam Rosenfield
236k 64 358 484

```
typedef std::map<size_t, some_class*> TMap;
TMap Map;
.......

for( TMap::const_reverse_iterator It = Map.rbegin(), end = Map.rend(); It != end; It++ )
{
    TMap::const_iterator Obsolete = It.base();   // conversion into const_iterator
    It++;
    Map.erase( Obsolete );
    It--;
}
```

answered Oct 18 '11 at 15:05
Nismo
11 1

If you don't need to erase everything as you go along, then to solve the problem, you can use the erase-remove idiom:

```
m_CursorStack.erase(std::remove(m_CursorStack.begin(), m_CursorStack.end(), pCursor),
m_CursorStack.end());
```

`std::remove` swaps all the items in the container that match `pCursor` to the end, and returns an iterator to the first match item. Then, the `erase` using a range will erase from the first match, and go to the end. The order of the non-matching elements is preserved.

This might work out faster for you if you're using a `std::vector` , where erasing in the middle of the contents can involve a lot of copying or moving.

Or course, the answers above explaining the use of `reverse_iterator::base()` are interesting and worth knowing, to solve the exact problem stated, I'd argue that `std::remove` is a better fit.

answered Sep 6 '11 at 13:08
gavinbeatty
146 11

Just wanted to clarify something: In some of the above comments and answers the portable version for erase is mentioned as (++i).base(). However unless I am missing something the correct statement is (++ri).base(), meaning you 'increment' the reverse_iterator (not the iterator).

I ran into a need to do something similar yesterday and this post was helpful. Thanks everyone.

answered Sep 21 '12 at 17:04
user1493570
27 6